VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Nástroj pro návrh a testování automatů

Tool for Design and Verification of Automats

2015

Daniel Vjačka

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Daniel Vjačka**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Nástroj pro návrh a testování automatů
Tool for Design and Verification of Automata

Zásady pro vypracování:

Cílem je mít nástroj na zkoušení návrhu automatů pro předmět UTI (deterministických/nedeterministických konečných automatů a Turingůvých strojů). Bude se skládat ze dvou nezávislých, ale propojitelných součástí.

1. Navrhněte textový formát (s částečným ohledem na čitelnost člověkem) pro přenos zmíněných automatů mezi oběma součástmi implementace.
2. Navrhněte a implementujte první součást - vizuální návrhář automatů pro webové prohlížeče s pomocí technologií HTML5 a JavaScriptu. Tento by měl být funkční ve verzích prohlížečů Firefox a Chrome aktuálních k datu odevzdání práce. Návrhář bude umožňovat vytvoření automatu dle učebnho textu UTI a přenesení automatu v textovém formátu.
3. Navrhněte způsob integrace návrháře do jiných webových aplikací tak, aby bylo jej možné vložit do vyhrazeného prostoru libovolné webové stránky a následně předat vytvořený automat JavaScriptovému kódu této stránky.
4. Navrhněte parametrizovaný algoritmus pro přibližné vyhodnocení správnosti (tedy, že kontrolovaný automat s nějakou úrovní jistoty funguje dle zadání). Při návrhu zvažte jaké vstupy a výstupy budou nejpraktičtější z hlediska testování schopností studentů.
5. Vytvořte ukázkovou webovou aplikaci, ve které naimplementujete algoritmus z bodu 4. a integrujete návrhář tak, aby bylo možné si vytvořit a otestovat automat. Popište postup pro zprovoznění aplikace včetně vložení vlastních dat.

Seznam doporučené odborné literatury:

[1] Jančar, Petr: Úvod do teoretické informatiky – učební text. Ediční středisko VŠB-TUO. 2007.

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. Jakub Macek**

Datum zadání: 01.09.2014

Datum odevzdání: 07.05.2015

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 7. května 2015 .....

Při práci na této bakalářské práci bych chtěl poděkovat mému vedoucímu Ing. Jakubovi Mackovi za jeho pomoc, za jeho obrovskou trpělivost a také bych rád poděkoval všem co mě při této práci podporovali.

Abstrakt

Cílem této bakalářské práce je vytvořit nástroj pro návrh a zkoušení konečných automatů a Turingových strojů pro předmět Úvod do teoretické informatiky. Tento nástroj by měl být vytvořen za použití technologií JavaScriptu, CSS, HTML a pro vytvoření algoritmu by měla být využita technologie C#.

Textová část bakalářské práce je rozdělena na tří hlavní části. První popisuje základní informace o použitých technologiích. Druhá část vysvětluje základy konečných automatů a Turingových strojů. A třetí část slouží k popsání vytvořeného programu.

Klíčová slova: JavaScript, CSS, HTML, C# ,UTI, konečné automaty, Turingové stroje, bakalářská práce

Abstract

The aim of this bachelor's thesis is creating a designer and tester for finite state automats and Turing machines for the purposes of the Introduction to Theoretical Computer Science course. The designer should be created using JavaScript, CSS and HTML technologies. The tester should be created using C# programming language.

The text part of this bachelor's thesis is divided into three main parts. First part describes basic information about used technologies. Second part explains basics of finite state automats and Turing machines. And the third part describes the user interace and the code behind created designer and tester tools.

Keywords: JavaScript, CSS, HTML, C# ,UTI, finite state machines, Turing machine, bachelor thesis

Seznam použitých zkratek a symbolů

HTML	–	Hyper Text Markup Language
Opensource	–	Systém volně dostupný pro úpravy
UTI	–	Úvod do teoretické informatiky
CSS	–	Cascading Style Sheets
DKA	–	Deterministický konečný automat
NKA	–	Nedeterministický konečný automat

Obsah

Seznam tabulek

Seznam obrázků

Seznam výpisů zdrojového kódu

1 Úvod

Tato bakalářská práce se týká předmětu Úvod do teoretické informatiky, dále pouze UTI. Tento předmět nás seznámí s výrokovou logikou, jazyky, regulárními výrazy a také konečnými automaty. V této práci se nadále budu zabývat pouze konečnými automaty, kdy vytvořím nástroj pro návrh a testování.

Tento nástroj by měl sloužit studentům k domácímu studiu, kdy si vytvoří v návrháři vlastní automat a následně si otestují jeho funkčnost. Tímto si studenti vyzkouší jestli danou problematiku dostatečně pochopili. Tento nástroj bude přizpůsoben tomu, aby jej nebylo možno zneužít při testech.

Bakalářská práce je vyhotovená tak, aby po přečtení této práce věděl čtenář základní informace a porozuměl danému problému. V první části budu popisovat základy mnou používaných technologií. Jsou zde vysvětleny základy těchto programovacích jazyků. Pro případné základní pochopení kódu naprogramovaného nástroje. Druhá část je zaměřena na vysvětlení a pochopení automatů a Turingových strojů. Popis těchto automatů bude od definice po názornou ukázku grafu automatů. Čtenář by měl při začátku třetí části rozumět základům programovaní a také funkčnosti automatů. V třetí části bude uživateli vysvětlen nástroj z pohledu programátora a nastíněny problémy, které se musely řešit v průběhu programování.

Celá aplikace je rozdělena na dvě částí, kdy pro přenos mezi částmi byl vytvořen vlastní textový formát. Tyto dvě části jsou vzájemně propojitelné a na sobě nezávislé.

První slouží k navrhnutí automatu, tato část je vytvořena za pomocí technologií JavaScript, CSS a HTML, pro vykreslovaní použití elementu canvas. Celá první část je vytvořena tak, aby byla možná jednoduchá integrace celého návrháře do webové stránky. Ovládání tohoto designéra je velice intuitivní.

Druhá část je využívána pro logiku programu. Zde se řeší převod z textového formátu do objektů. Poté je zde umístěn celý algoritmus na testování a průběh automatů.

2 Používané technologie

Po krátkém úvodě, si v krátkosti předvedeme používané technologie. Ke každé technologii si uvedeme krátký popis a ukázku. Mezi nejvíce používané technologie patří JavaScript a C#. Dále jsem také samozřejmě musel využit i jiných technologií jako například HTML,CSS atd. Z této kapitoly se čtenář nestane programátorem, ale může pochopit základní informace. Pokud čtenáře zaujme nějaká technologie, může si na internetu najít veškeré informace a zjistit si detaily týkající se daného problému. Vybrané a zajímavé problémy, které budou možná trochu složitější a obsáhlejší rozeberu v třetí částí mé bakalářské práce, kde se budu zabývat celým programem a postupem programátora.

2.1 HTML

Jako první budu psát o jazyku HTML. HTML neboli HyperText Markup Language se používá pro tvorbu internetových stránek. Prohlížeče poté rozluští tento kód a převedou ho do zobrazitelné formy. Níže je uvedená základní konstrukce html souboru.

Jako hlavní tag, což je základní konstrukce tohoto jazyka, který se uvádí v hranatých závorkách je <html>. Tyto tagy se používají pro vymezení vzhledu a konstrukce stránky. Dále zde máme rozdělení na hlavičku neboli <head> a tělo stránky neboli <body>, což jsou také párové tagy, takže se musí ukončit pomocí stejného tagu s lomítkem </body>. Tagů může html soubor obsahovat opravdu hodně, ale snažím se nastínit pouze základní povědomí o technologiích. Pomocí těchto tagů můžeme taky provádět spolu s CSS celkový vzhled stránky, který si budeme vysvětlovat později v samostatné podkapitole.

```
<html>
    <head>
        <title>Ukazka</title>
    </head>
    <body>
        <p>Odstavec s textem</p>
    </body>
</html>
```

Výpis 1: Struktura v HTML

2.1.1 Canvas element

Spolu s uvedením HTML5 přišel i canvas element, který využívám při mé bakalářské práci. Je využíván při vykreslování grafiky na internetové stránce. Pouze při spojení s technologií javascript je schopen vykreslovat grafické prvky.

Tento element jsem použil jako návrhář pro vykreslování konečných automatů. V tomto elementu jsem pomocí javascriptu dovedl vykreslit uzly a přechody pomocí šipek. Canvas je velká součást první části mého programu a řeší veškerou grafiku.

Níže je uvedená struktura tohoto elementu v kódu, nejdůležitější atributy jsou ID, width a height. Pomocí ID atributu se v javascriptu budeme odkazovat na tento element

a můžeme s tím pracovat. Jako další nastavujeme výšku a šířku oblasti pro vykreslování grafiky. Obsahuje taky i další atributy, ale ty v tuto chvíli nejsou tak důležité.

```
<canvas id="canvas" width="800" height="600" > </canvas>
```

Výpis 2: Canvas element v HTML

2.2 JavaScript

Jako další programovací jazyk uvedeme JavaScript. Tento jazyk je využívaný spolu s HTML, je využíván například pro ověřovaní vstupních dat, řešení problémů na straně prohlížeče atd. JavaScript je v této době hojně využíván a dokáže vytvořit dynamickou internetovou stránku, která se vkládá přímo do HTML. Způsoby vložení scriptu jsou dva. První je přímé vložení do HTML kódu a druhý způsob je pomocí připojení externího souboru. Více si ukážeme v krátké ukázce. Můj javascriptový soubor je vložený externě a obsahuje všechny potřebné informace pro návrhář. Vytvořené soubory mají přípony .js nebo .jse.

```
<script>
    alert (" Interni zapis JavaScriptu s vyskakovacim okenkem");
    document.write("Zapis pomoci skriptu na obrazovku");
</script>

<script type="text/javascript" src="js/externisoubor.js"></script>
<!-- Externi skript -->
```

Výpis 3: Úvod do JavaScriptu

V této krátké ukázce jsme si ukázali implementaci interního a externího zápisu JavaScriptu. Základem jsou párové atributy <script>, a u externího zápisu také atributy určující zdroj javascriptu, typ atd. Další atributy jsou v nejnovějších prohlížečích zbytečné.

V interním zápisu jsme si uvedli dva základní příkazy. Je nutné dát pozor na velká a malá písmena, JavaScript je CaseSensitive a příkaz Alert není totéž co alert. Příkaz alert znamená oznamovací okénko, které obsahuje text v závorkách. Další uvedený příkaz je document.write, který umožňuje přímý zápis textu do prohlížeče.

Samotný jazyk je velmi složitý, a pro úvod bude stačit jednoduchý základ.

2.3 CSS

Jak jsme se už u HTML zmínili tak CSS jazyk je navržen k formátování stránky. CSS neboli kaskádové styly (v angličtině Cascading Style Sheets) používá pro práci s designem stránky tag <style> nebo také atribut style. K úpravě, ale také můžeme použít tagy jako je <p> pro odstavec <h1> pro nadpis a tak dále. Uvedeme si jednoduchou syntaxi pro představu.

```
<html>
  <head>
```

```
<link rel="stylesheet" type="text/css" href="css/externisoubor.css"/>
  </head>
  <body>
    <h1> Zeleny nadpis</h1>
    <p style="color:red">Cerveny odstavec</p>
    <style>
        h1{color: green}
    </style>
  </body>
</html>
```

Výpis 4: Ukázka CSS kódu

Z tohoto kódu můžeme vyčíst, že existují tři způsoby vložení CSS kódu. U odstavce, což je atribut <p>. Další, méně používaný způsob formátování je pomocí tagů style a zde si určíme který element chceme upravovat. V tomto případě jsme si určili u nadpisu zelenou barvu. Jako poslední a nejpoužívanější způsob je pomocí externího souboru. Kde zápis vypadá stejně jako u našeho tagu style. V mé bakalářské práci využívám CSS soubor minimálně, ale uvedl jsem zde krátkou podkapitolu pro uvedení všech mnou používaných technologií.

2.4 C# a Visual Studio

Poslední z uvedených programovacích jazyků bude C#. Tento jazyk je využíván pro druhou část tohoto programu, tedy pro rozparsování informací do objektů a pro výpočet algoritmu konečných automatů a Turingových strojů. Je to objektově orientovaný programovací jazyk, který vychází z Javy a jazyku C. Je konstruován tak, aby byl jednoduchý k pochopení, moderní a víceúčelový. Opět si uvedeme pár řádku kódu pro představu a objasnění konstrukce.

```
static void Main(string[] args)
    {
        int a = 5;
        Console.WriteLine(a);
        Console.ReadKey();
    }
```

Výpis 5: Základ C#

K celé druhé části využívám .NET framework 4.5, prostředí které pro toto používám je Visual Studio Ultimate 2013. Přesněji řečeno využívám ASP.NET což je webový framework, který se využívá pro tvorbu webových aplikací.7 10 Po pár větách neznámých slov se můžeme cítit zmateni, ale v krátkosti vysvětlím o co jde. Prostředí o kterém mluvím, je vývojové prostředí, neboli program ve kterém píšeme zdrojový kód. Poté se program postará o zkompilování a spuštění programu. Framework je zjednodušeně řečeno sada knihoven, které využíváme ke spuštění programu a zajištění aby fungovalo vše jak mělo.

Poslední věc kterou zmíním a která mě zaujala je, že Visual Studio při vytváření ASP.NET aplikací využívá Twitter Bootstrap framework.9 Tento framework je volně dostupný na internetu a je open source. Je využíván pro jednoduchý, rychlý a opravdu

elegantní vzhled webu. Jelikož jsme na konci kapitoly a máme základní přehled o technologiích, tak můžeme říci, že tento framework využívá od HTML po CSS prvky. Tyto částí kódu je možno velmi jednoduše implementovat na naší internetovou stránku či webovou aplikaci. Pomocí tohoto frameworku, který nabízí opravdu mnoho prvků a elementů vypadá internetová stránka hned na první pohled vcelku dobře. Sám tento framework využívám při programování webů a myslím si, že je velmi užitečný. Pomocí frameworku je možno vytvořit dokonce responsivní web bez větších či pokročilejších znalostí programování. Nyní si opět předvedeme ukázku a implementaci tohoto frameworku.

```
<!DOCTYPE html>
<html>
  <head>
    <title>Bootstrap vzor</title>
    <script src="http :// code.jquery.com/jquery-latest.js"></script>
        <script src="https :// code.jquery.com/jquery-1.11.1.js"></script>
        <!-- Bootstrap -->
        <link rel="stylesheet" href="https :// maxcdn.bootstrapcdn.com/bootstrap/3.3.1/css/
            bootstrap.min.css">
        <link rel="stylesheet" href="https :// maxcdn.bootstrapcdn.com/bootstrap/3.3.1/css/
            bootstrap-theme.min.css">
  </head>
  <body>
    <p class="text-warning">Etiam porta sem malesuada magna mollis euismod.</p>
    <div class="jumbotron"></div>
    <script src="https :// maxcdn.bootstrapcdn.com/bootstrap/3.3.1/js/bootstrap.js"></script>
  </body>
</html>
```

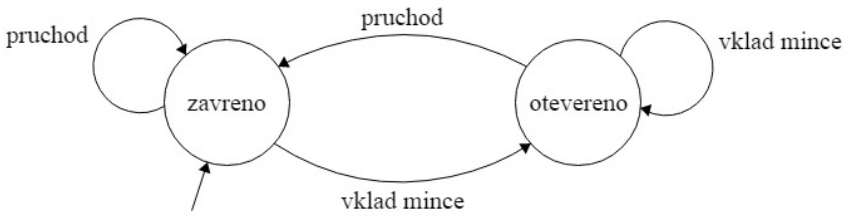
Výpis 6: Úvod do Twitter Bootstrap

Toto je funkční kód twitter bootstrap, nesmíme zapomenout na nejnovější verze jquery a bootstrap. Z CSS a js souborů totiž bereme informace o tom, jak má daný element vypadat čehož si všimneme u formátování textu v párových atributech <p> a . Pomocí atributu *class* určujeme vzhled.

Po krátkém vysvětlení programovacích jazyků a používaných technologií, se můžeme vydat do vysvětlování a pochopení automatů a Turingových strojů.

3 Automaty

Konečný automat (zkráceně KA, anglicky Finite state machine) je abstraktní model hojně využívaný v teoretické informatice. Používá se pro rozpoznávání jazyků, neboli pro studium formálních jazyků. Tento model také můžeme rozpoznat v automatu na kávu. Nebo další případ, který znázorníme graficky je turniket na průchod bránou. Ve chvíli kdy přicházíte se turniket nachází ve stavu zavřeno. V tomto stavu setrvává dokud nevhodíte minci. Můžeme si v grafu automatu povšimnout přechodu do stavu otevřeno pomocí vložení mince. Ve stavu otevřeno setrvává i po vložení mince, zpět do stavu zavřeno se dostaneme pomocí přechodu, neboli průchodu bránou. A pokud se pokusíme znovu projít tak neprojdeme, jelikož se stále nachází ve stavu zavřeno.



Obrázek 1: Ukázka automatu

Je důležité zdůraznit, že automat pracuje s konečným počtem stavů a také konečným počtem vstupních podnětů, neboli vstupní abecedou. Stav automatu je ovlivněn vnějšími podněty, kdy při vstupním podnětu automat přesně ví do kterého stavu se dostane.

K popisu automatů se používají také jiné způsoby a to buď stavovým diagramem, neboli grafem automatu. Dále se přechodové funkce dají popsat i takzvanou tabulkou přechodů. Tyto způsoby se používají pro jednodušší a srozumitelnější pochopení automatu.

Graf automatu se skládá z konečné množiny stavů, který je reprezentován pomocí koleček, neboli uzlů. V automatu můžeme také nalézt dvojitá kolečka, která reprezentují konečný stav automatu. Z koleček - stavů vycházejí šipky, které reprezentují přechody mezi stavy. Pokud šipka směřuje do stejného uzlu, tak po přečtení písmene zůstává automat ve stejném stavu. Tyto přechody reprezentují naše vnější podněty. Počáteční stav bývá vyjádřen šipkou, která míří k počátečnímu stavu $\rightarrow$.

Další způsob vyjádření je tabulka přechodů. Tato tabulka může být srozumitelnější pro zápis složitějších automatů. Tabulka je vytvořena tak, že v záhlaví sloupců máme množinu vstupních prvků a v záhlaví řádků máme množinu stavů. Začáteční stav je značen symbolem šipky doprava $\rightarrow$ a konečný stav reprezentujeme symbolem šipky doleva $\leftarrow$.

Poslední způsob zobrazení je stavový strom. Tento způsob zápisu budeme nejčastěji využívat u nedeterministických automatů. Tento strom nám totiž může ukázat způsob

průchodu celým automatem a to i v případě pokud nastane ve stavu případ, kde bychom měli dvě možnosti přechodu. Pro laiky je tento způsob zápisu také lépe pochopitelný a srozumitelný. Způsobem nákresu připomíná binární strom, kdy se navíc přidávají po stranách přechody se vstupem, abychom přesně věděli, který znak momentálně automat přečetl. Tento stavový strom je k náhledu v kapitole Nedeterministických automatů a zde je také vysvětlen popis průchodu stromem.

Konečné automaty se dají dělit na dvě skupiny. A to na deterministické konečné automaty dále už jenom DKA a nedeterministické konečné automaty zkráceně NKA. Rozdíl mezi těmito automaty je minimální, a lze jednoduše převést NKA na DKA. Pouze pro představu, NKA se od DKA liší například v tom, že v daném stavu má více přechodů pro jeden vstupní podnět. Detailněji se o daných typech budu zabývat v nadcházejících kapitolách.

Určitě by byla škoda nezmínit také abstraktní výpočetní model, související s mou bakalářskou prací a automaty. A to je Turingův stroj. Tento stroj se používá hlavně pro určení složitosti algoritmu a pro určení složitosti problému. Je to konečný automat, který pracuje se vstupní páskou, kde čte vstupní symboly a v přechodové funkci má uloženou funkcionalitu stroje.

3.1 Deterministické

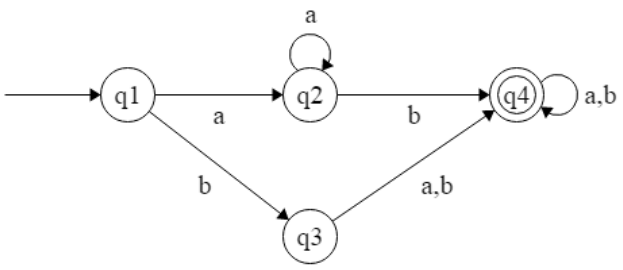
Deterministický konečný automat je jinak řečeno jednoznačný. Tento automat je uzpůsoben tak, že má pouze jeden počáteční stav. U tohoto automatu také platí pravidlo, že musí být jasně a jednoznačně dána cesta pro každý možný vstupní symbol ze vstupní abecedy. To znamená, že z každého uzlu automatu, což jsou všechny možné stavy, musí být stanoveny cesty pro celou vstupní abecedu. Automat může mít více přijímacích stavů, pokud bude rozpoznán jazyk tak automat skončí v jednom z konečných stavů. Tento odstavec můžeme shrnout do následující krátké definice. 1

Definice 3.1 *Deterministický konečný automat (zkráceně DKA) je každá pětice prvků* $A=(Q,\sum,\delta,q_0,F)$, *kde*

- Q *znamená konečnou neprázdnou množinu stavů,*
- $\sum$ *znamená konečná neprázdná množina vstupních prvků, tzv. vstupní abeceda*
- δ *je přechodová funkce -* $Qx\sum$ -> Q
- q_0 *je počáteční neboli iniciální stav -* $q_0 \in Q$
- F *znamená množina koncových neboli přijímacích stavů,* $F \subset Q$

Pro praktické osvětlení DKA jsme si navrhli jednoduchý čtyřstavový automat, který přijímá slova nad vstupní abecedou {a,b}. Jeden z výrazů, které tento automat přijme je "aaabab", pro který si následně také vysvětlíme simulaci tohoto automatu.

V grafickém znázornění, jak jsme si vysvětlovali, vypadá automat takto.



Obrázek 2: Deterministický konečný automat - stavový diagram

Toto grafické znázornění definujeme jako automat takto, A =(Q,$\sum$,δ,q_0,F), kde Q = {q1,q2,q3,q4}, $\sum$ = {a,b}, q_0 = q1, F = {q4}, a přechodová funkce je definována takto δ(q1,a)=q2, δ(q1,b)=q3, δ(q2,a)=q2, δ(q2,b)=q4, δ(q3,a)=δ(q3,b)=q4, δ(q4,a)=δ(q4,b)=q4.

V této definici máme všechny potřebné informace pro práci s automatem.

Nyní si popíšeme průběh DKA. Pokud na vstupu bude výraz "aaabab", tak začínáme v počátečním stavu q1. Slovo čteme zleva doprava. Po přečtení prvního písmene "a"se přesune podle šipky do stavu q2 a ze slova se odebere první písmeno "a"a tím pádem zůstane "aabab". Dále se ze slova přečte písmeno "a", kdy zůstáváme ve stavu q2. Další přečtené písmeno je opět písmeno "a", kdy opět zůstaneme ve stejném stavu. Nyní zůstane pouze "bab", kdy při přečtení písmena "b"se přesuneme do konečného stavu q4. Toto je konečný stav, ale ještě není přečteno celé slovo, takže automat ještě neskončil. Povšimneme si, že v konečném stavu máme smyčku pro oba možné vstupy "a,b". Ve slově nám zůstalo už pouze "ab", takže po přečtení slova víme, že automat, vzhledem k přechodu typu smyčka, zůstane v q4. A tímto je přečteno celé slovo, automat skončil v konečném stavu a tímto je automat platný. 6

Zde máme ukázku tabulky přechodů, která vychází z předchozího grafu. Můžeme si povšimnout, že počáteční stav q1 je značen šipkou $\rightarrow$ a konečný stav je značen $\leftarrow$. Záhlaví sloupců nám tvoří vstupní abeceda, tedy {a,b}.

	a	b
$\rightarrow q_1$	q_2	q_3
q_2	q_2	q_4
q_3	q_4	q_4
$\leftarrow q_4$	q_4	q_4

Tabulka 1: Deterministický konečný automat - přechodová tabulka

3.2 Nedeterministické

Nedeterministický konečný automat se od deterministického mírně liší, u DKA jsme měli jasně nadefinovány přechody a bylo nám jasné, co DKA zrovna provede. NKA se liší v

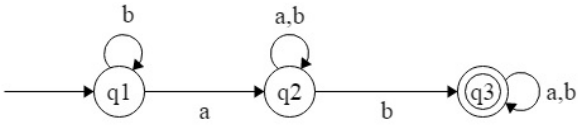
tom, že na rozdíl od DKA, může mít více počátečních stavů. Což v definici také znamená, že to bude množina stavů. Další markantní rozdíl mezi automaty je v tom, že NKA může mít z jednoho stavu více přechodů pro jeden znak, v závislosti na vstupní abecedě, nebo také žádný. Při vlastnosti automatu více přechodů podle stejného vstupního prvku může nastat rozvětvení algoritmu. V tomto případě končí automat, pokud alespoň jedna větev skončí v konečném stavu. Jelikož je to konečný automat, tak určité prvky definice zůstávají stejné. Jako DKA, kdy může mít žádný nebo více koncových stavů.

Opět si shrneme všechny informace do krátké definice. 1

Definice 3.2 *Nedeterministický konečný automat (zkráceně NKA) je každá pětice prvků A =(Q,∑,δ,I,F), kde*

- *Q znamená konečnou neprázdnou množinu stavů,*
- *∑ znamená konečná neprázdná množina vstupních prvků, tzv. vstupní abeceda*
- *δ je přechodová funkce - Qx∑ -> P(Q)*
- *I je počáteční neboli iniciální stav - I⊂Q*
- *F znamená množina koncových neboli přijímacích stavů, F⊂Q*

Pro názorné vysvětlení definice a celého NKA jsme navrhli stručný třístavový automat. Tento automat přijímá jako vstupní abecedu {a,b}. Jeden z výrazů, který tento automat přijímá a pro který si provedeme simulaci je "baabbb". Graf automatu, neboli stavový diagram bude vypadat takto.



Obrázek 3: Nedeterministický konečný automat - stavový diagram

Pro srovnání s DKA zde uvedu také definici tohoto automatu. Kdy A=(Q,∑,δ,I,F), kde Q = {q1,q2,q3}, ∑ = {a,b}, I = {q1}, F = {q4}. Přechodovou funkci si definujeme takto δ(q1,a)=q2, δ(q1,b)=q1, δ(q2,a)=q2, δ(q2,b)={q2,q3}, δ(q3,a)=(q3,b)=q3.

A teď můžeme popsat grafické znázornění NKA a taky průběh automatu pokud, se vyskytne na vstupu slovo "baabbb". Ocitáme se na počátečním stavu a to q1. Ze slova čteme první znak a to "b"kdy si můžeme povšimnout smyčky a proto zůstáváme ve stavu q1. Následující znak je "a", kdy se pomocí přechodu posouváme do stavu q2. A opět ze slova přečteme písmeno "a", kdy zůstáváme stále ve stavu q2. Nyní bychom mohli říci, že nastává problém, máme pro písmeno "b"které se ocitá na vstupu dvě možnosti. My si pouze vybereme jednu cestu a přejdeme do stavu q3. Ve stavu q3 už máme smyčku pro stavy "a,b"a proto jsme v konečném stavu a automat je platný. Kdybychom si vybrali ve

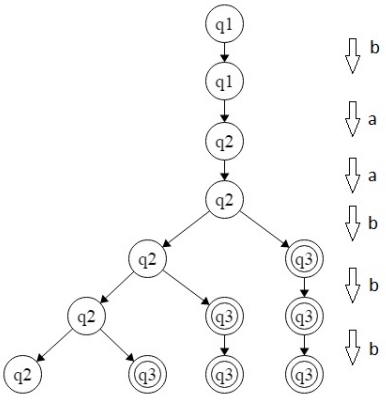
stavu q2 druhou možnost a to možnost smyčky, tak bychom po přečtení zbytku písmen byli stále v q2 a tento stav není konečný. Takže slovo bychom touto volbou nepřijali.

U této tabulky přechodů si můžeme povšimnout speciálního zápisu. Jak jsme si vysvětlovali, tak NKA může obsahovat více přechodů pro stejný znak či pro daný znak nemusí být nadefinován žádný přechod. V této tabulce je počáteční stav opět značen šipkou při stavu q1 a koncový ve stavu q3. Změna oproti DKA zde nastává ve stavu q2, kde při přechodu pomocí vstupu "b"máme množinu dalších stavů a to $\{q_2,q_3\}$. Což znamená, že z tabulky můžeme vyčíst, že ze stavu q2 se dostaneme pomocí vstupu b zpět do stavu q2 a poté do stavu q3, což je stav koncový.

	a	b
$\rightarrow q_1$	q_2	q_1
q_2	q_2	$\{q_2,q_3\}$
$\leftarrow q_3$	q_3	q_3

Tabulka 2: Nedeterministický konečný automat - přechodová tabulka

Jako poslední grafické vyjádření automatu si ukážeme stavový strom. 8 Na tomto grafickém vyjádření lze nejlépe pochopit průběh celého automatu. Můžeme si totiž povšimnout kudy se automat může pohybovat. Máme znázorněný celý průběh čtení slova a možné odbočky. Zde je vytvořený stavový strom pro náš třístavový nedeterministický automat. Jak jsme už řekli, přechod do stavu q2 nebyl problém a průběh byl velmi podobný DKA. Ale u stavu q2 nastal problém v tom, že automat měl možnost si vybrat mezi více přechody. Ve stavovém stromu si můžeme povšimnout, že u stavu q2 máme znázorněny dvě cesty. Jedna vede zpět do q2 a druhá do q3 což je konečný stav. Poté máme opět na vstupu znak "b", což znamená, že se scénář opakuje. Může si vybrat jít do q3 nebo q2. Takto bude probíhat scénář až do konce vstupního slova. Pokud se podíváme důkladně na strom, tak si všimneme že i po dvou přečtených písmen "b"se stále můžeme dostat do konečného stavu a automat by byl platný a slovo prošlo.



Obrázek 4: Nedeterministický konečný automat - stavový strom

4 Turingové stroje

Tento stroj je velmi silná výpočetní jednotka, která dokáže vypočíst vše co reálný počítač. Poprvé byl představen matematikem Alanem Turingem v roce 1936.

Tento model se velmi podobá konečným automatům, ale je mnohem výkonnější a také má neomezenou paměť. Tyto stroje se proto můžou používat k vyřešení složitějších problémů. Turingova neomezená paměť, je realizována pomocí takzvané pásky, která může být za určitých okolností nekonečná. Čímž je myšleno, že se může neustále doplňovat a stane se nekonečnou. Tato páska obsahuje vstupní řetězec a zbytek pásky jsou prázdná místa, takzvaná blank spaces. Další vlastnost Turingova stroje, kterou se liší od konečných automatů je, že kromě čtení je možnost na pásku zapisovat a pohybovat se po pásce oběma směry. Čtecí hlava má taky možnost stát na místě. Pohybování hlavy po pásce pokračuje do doby, než dosáhne konečného stavu. 2

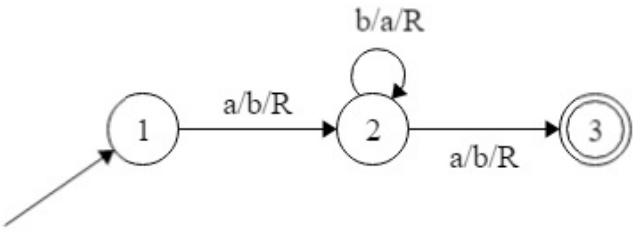
Pohyby čtecí hlavy, zápis a čtení jsou realizovány pomocí přechodů. Tímto si můžeme naprogramovat vlastní Turingův stroj, který bude vykonávat naší funkcionalitu. Pomocí programování našeho stroje můžeme reprezentovat jakýkoliv programovací jazyk.

Níže je opět napsaná definice 1 tohoto stroje:

Definice 4.1 *Turningový stroj je každá šestice prvků A =(Q,Γ, s, b, F, δ), kde*

- *Q znamená konečnou neprázdnou množinu stavů*
- *Γ znamená konečná neprázdná množina vstupních symbolů a znaků*
- *s∈Q znamená počáteční stav*
- *b∈ Γ je symbol, který reprezentuje prázdný symbol (b není součástí vstupní abecedy)*
- *F⊂Q znamená množina koncových stavů*
- *δ: Q×Γ → Q×Γ× {-1,0,+1}*

Nyní si pro ukázku uvedeme také grafické znázornění Turingova stroje a jeho přechodových funkcí.



Obrázek 5: Turingův stroj

Takto by měl poté v návrháři vypadat námi navržený Turingův stroj. Kdy u textu přechodů máme definovanou přechodovou funkci a máme zde jeden počáteční stav a

také jeden stav konečný. Tento stroj je velmi primitivní, ale v návrháři bude možnost navrhnout stroje složitější.

Jako u každého modelu si uvedeme funkčnost a průchod tímto strojem. Opět uvedu formální zápis tohoto Turingova stroje. Kdy A=(Q,Γ, s, b, F, δ), kde Q={1,2,3}, Γ = {a,b}, s=1, b=B, F={3}. Přechodovou funkci Turingova stroje si definujeme takto δ(1,a)=(2,b,R), δ(2,b)=(2,a,R), δ(2,a)=(3,b,R).

Nyní po zapsání formální definice si popíšeme průběh tohoto stroje. Vstupní řetězec budeme mít zadán jako "abba", kdy by se nám měl první a poslední znak pokud bude "a"změnit na znak "b"a současně znaky "b"změnit na znaky "a". Čímž by měl výstupní řetězec mít tvar "baab". Při vstupu řetězce se nám čtecí hlava nachází nad znakem "a"a nacházíme se ve stavu 1. Zde máme pro znak "a"nadefinovanou přechodovou funkci. Tato přechodová funkce nám říká, že po přečtení znaku "a"ho přepiš znakem "b"a čtecí hlavu posuň o jeden znak doprava. Výstupní řetězec v tuto chvíli je "bbba". Nyní se nacházíme ve stavu 2 a čtecí hlava je na druhém znaku "b". Opět zde máme pro znak "b"nadefinovanou přechodovou funkci. Znak "b"se nám přepíše znakem "a"a čtecí hlava se nám opět posune o jedno pole doprava. Nyní ale zůstáváme stále ve stavu 2, kdy opět čteme znak "b", pro který provedeme stejnou operaci. Nyní náš výstupní řetězec má tvar "baaa". Hlava se nyní nachází nad znakem "a"a ve stavu 2 máme pro znak "a"nadefinovanou přechodovou funkci, pomocí které se dostaneme do konečného stavu. Přechodová funkce nám definuje, že při přečtení znaku "a"zapíše znak "b"a posune se doprava. Nyní je naše slovo celé přečtené Turingovým strojem a my se nacházíme v konečném stavu. Výsledný řetězec je tedy "baab". 4

5 Popis aplikace

Po teoretické části se vydáme do části poslední a to praktické, kde si ze začátku probereme návrh aplikace, prostředí. Dále se budeme věnovat samotné implementaci a konečnému grafickému prostředí a testování aplikace s reálnými daty. Zadání mé bakalářské práce bylo sestrojit nástroj na testování automatů. Tento nástroj by měl být schopen vytvořit grafický návrh a dle algoritmu vyhodnotit správnost automatů. Celou dobu bych se měl řídit skripty z předmětu UTI. V předmětu jsme se zaobírali automaty a k tomu jsem měl zadáno vytvoření algoritmu pro testování Turingových strojů. Kompletně by měl být program schopen vyřešit tyto tři modely:

- Deterministické automaty
- Nedeterministické automaty
- Turingovy stroje

Tato aplikace by měla sloužit studentovi k domácímu studiu, proto bude navrhnuto grafické prostředí, zadávání vstupů a výstupů, s ohledem na domácí studium. Student, který zatím není v tomto druhu látky znalý, by měl mít možnost si zkusit navrhnout automat a vyzkoušet jeho funkčnost. Pomocí jednoduchého ovládání si student může automat jakkoliv předělat a během krátké chvíle by měl mít možnost vytvořit zcela nový automat.

Celá aplikace, jak jsem zmínil, se skládá ze dvou částí. Tyto částí by mezi sebou měli být nezávislé, ale propojitelné pomocí mnou navrženého textového formátu.

Při mnou navrženém textovém formátu bych měl zohlednit čitelnost člověkem. Proto pravděpodobně budu využívat pouze potřebné informace s jasně danými instrukcemi.

5.1 První část

Pro tvoření první části jsem měl jasně stanovené jaké technologie bych měl využít. Za úkol bylo vytvořit vizuální návrhář pro webové aplikace za použití technologií HTML a JavaScript. Při návrhu bych se měl zaměřit na to, jakým způsobem se s tím bude pracovat. Pro kreslící plochu jsem se rozhodl využít canvas element, který je využíván pro grafiku na webových stránkách. Zde jsem také musel zohlednit to, co by uživatel mohl chtít vytvořit a jaké funkce bude potřebovat. Z těch základních:

- nakreslit uzel
- poznačit koncový stav
- poznačit počáteční stav
- nakreslit přechod z uzlu do uzlu
- nakreslit smyčku přechodu

Z toho jsem usoudil, že bude nutno vytvořit objekty pro přechody a uzly. Nakonec bych neměl zapomenout do první části zakomponovat převod do textové části.

5.2 Druhá část

Při návrhu druhé části, což mělo řešit algoritmus správností automatu jsem si zvolil technologie sám. Osobně jsem si pro druhou část vybral jazyk C#. Uvažoval jsem nad implementací do ASP.NET technologie, která je mi blízká. V první řadě bych měl v tomto jazyku vyřešit převod textové části do objektů. Textový formát si musím převést do objektů, jelikož zde budu řešit algoritmus pro dané matematické modely. Přemýšlel jsem, že algoritmus pro vyřešení deterministického automatu by mohl být nejjednodušší a z něho by mohli vycházet další algoritmy pro řešení zbylých modelů.

Jako další krok nesmíme zapomenout na ošetření podmínek. Například deterministický automat nesmí obsahovat, jak jsme zmínili v teoretické části, více než jeden začátek. Všechny tyto podmínky bychom měli zachytit a dát uživateli vědět, zda nastala chyba a jaká chyba to je. Uživatel by měl možnost si chybu opravit a znova si zkontrolovat automat, dokud nebude mít automat správnou podobu.

5.2.1 Algoritmus DKA

Při popisu aplikace uvažujeme nad tím, jak by aplikace mohla pracovat. Z tohoto důvodu bychom si měli také uvést algoritmus, jak by nástroj mohl ověřit správnost aplikace a vyřešení automatu. Tento algoritmus je napsán pseudokódem a v implementaci si uvedeme jak tento pseudokód vypadá v aplikaci a zda funguje jak má.

Tento algoritmus už uvažuje nad tím, že automat už je naimplementován v objektech a je možné s ním pracovat a budeme zde ověřovat správnost automatu.

```
function AlgoritmusDKA(vstup)
– Kontrola automatu z matematickeho hlediska
IF pocet_startovnich_prvku != 1
   THEN Vypis chybovou hlasku neplatneho automatu
ENDIF
IF pocet_konecnych_prvku == 0
   THEN Vypis chybovou hlasku neplatneho automatu
ENDIF
FOREACH Uzly
   Kontrola jednotlivych prechodu
END FOREACH
– Kontrola vstupniho slova automatem
FOREACH Uzly
   IF je uzel startovni
      THEN uzel← startovni_uzel
   ENDIF
END FOREACH
FOREACH pismeno ze vstupniho slova
   FOREACH prechody v uzlu
      IF text prechodu je roven pismenu vstupniho slova THEN
         uzel ← nasledujici uzel z prechodu
         konec
      END IF
   END FOREACH
END FOREACH
– po pruchodu vsemi uzly vrati konec uzlu zda je true nebo false
```

```
RETURN konec uzlu
```

Výpis 7: Pseudokód k DKA

Zde je zjednodušený algoritmus pro kontrolu automatu z matematického hlediska a algoritmus pro průběh slova automatem.

5.2.2 Algoritmus NKA

Algoritmus pro nedeterministické by byl z velké části podobný. Stejně jako u deterministického automatu si musím zkontrolovat zda startovní a konečné stavy odpovídají matematické definici. Poté si najdeme startovní uzly, kdy jich může být více než jeden a pro každý uzel si zavoláme funkci pro průběh slova automatem. Nyní si pseudokódem ve zkratce ukážeme, jak může vypadat funkce pro vypočtení výsledku průběhem slova automatem.

```
function AlgoritmusNKA(vstup,uzel)
– posledni uzel v kterem se nalezame a vrati true nebo false
IF vstupni retezec prazdny
   THEN RETURN uzel.konec
END IF
IF prechody v uzlu neobsahuji pismeno ze vstupu
   THEN RETURN false
END IF
IF prechod v uzlu pro nasladujici pismeno je rovno 1
   THEN AlgortimusNKA(vstup_bez_prvniho_znaku,nasledujici_uzel)
END IF
IF prechod v uzlu pro nasledujici pismeno je rovno nebo vetsi 2 THEN
   FOREACH prechod z uzlu pro dane pismeno
      AlgortimusNKA(vstup_bez_prvniho_znaku,nasledujici_uzel)
   END FOREACH
END IF
– Vraceni vysledku
```

Výpis 8: Pseudokód k NKA

5.2.3 Algoritmus TNG

A jako poslední musíme řešit algoritmus pro vyřešení Turingova stroje. Opět si uvedeme zjednodušený pseudokód z důvodu rozsahu a složitosti kódu. Budeme řešit průběh slova strojem, tím pádem pohyby čtecí hlavy, čtení a zapisování. Začátkem se mírně podobá předchozím kódům, jelikož budeme muset najít startovní prvek. Poté procházíme uzly pomocí while smyčky, dokud nenarazíme na konečný uzel. Pseudokódem si uvedeme vnitřek tohoto cyklu.

```
FOREACH prechody v uzlu
IF prechod obsauje znak pro precteni THEN
   IF znak_pro_zapis != N ( znak N pro zanechani stejne hodnoty ) THEN
      IF znak_pro_zapis == B ( znak pro blank space) THEN
         vstupy[pozice] zapis prazdny znak
```

```
        ELSE vstupy[pozice] zapis znak_pro_zapis
        END IF
    END IF
    IF se chci posunout vlevo THEN
        IF pozice == 0 THEN
            presunuti celeho retezce o jednu pozici doleva
            vytvoreni na pozici 0 prazdny znak
        ELSE pozice--
    END IF
    IF se chci posunout vpravo THEN
        pozice++
        IF pozice >= delka vstupniho retezce
            vytvoreni na dane pozici prazdny znak
        END IF
    END IF
    pridani nasledujiciho uzlu z prechodu
END IF
END FOREACH
```

Výpis 9: Pseudokód k TNG

Po úspěšném skončení kód vrátí true a vím, že vstupní řetězec prošel Turingovým strojem. Nyní jsme si uvedli všechny tři možné algoritmy v pseudokódu a zaměříme se na návrh grafického prostředí. Poté budeme mít všechny potřebné informace pro naprogramování aplikace.

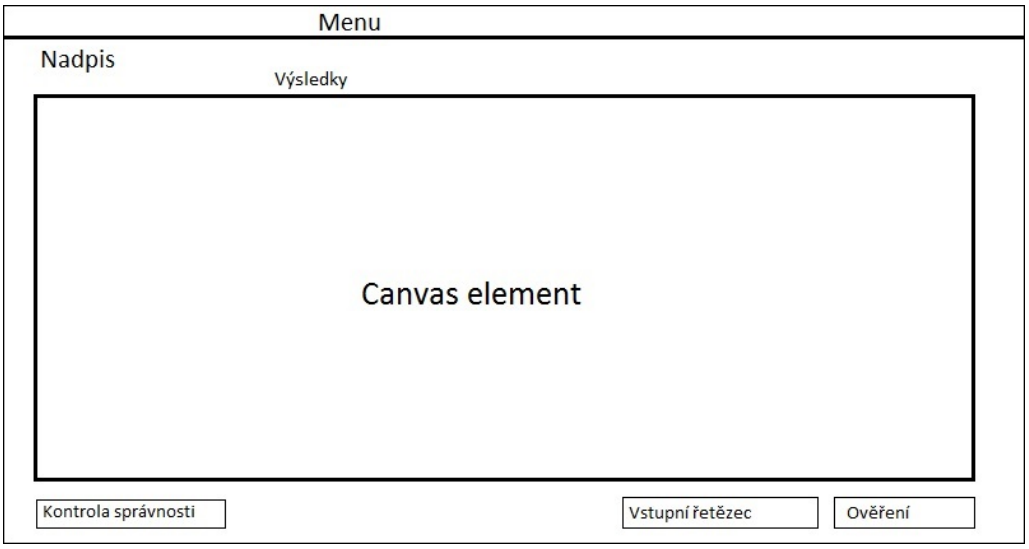
6 Grafické prostředí

6.1 Návrh grafické prostředí

Při vytváření návrhu grafického prostředí se musíme zamyslet nad tím, kdo bude aplikaci používat a pro koho by měla být uzpůsobena. Jelikož mám jako bakalářskou práci nástroj na testování automatů, tak největší část uživatelů budou tvořit studenti. Tohle si musíme uvědomit. Další část, nad kterou bychom se měli zamyslet, je které prvky budou na obrazovce. Při mé aplikaci to pravděpodobně budou

- tlačítko pro kontrolu automatu
- tlačítko pro kontrolu vstupního řetězce
- nadpis
- canvas element
- výběr automatu
- textBox pro vstupní řetězec
- výstup

Také bych měl zohlednit, aby pro kreslení automatů byla vyhrazena dostatečná plocha a aby všechny důležité prvky byly jasně viditelné.

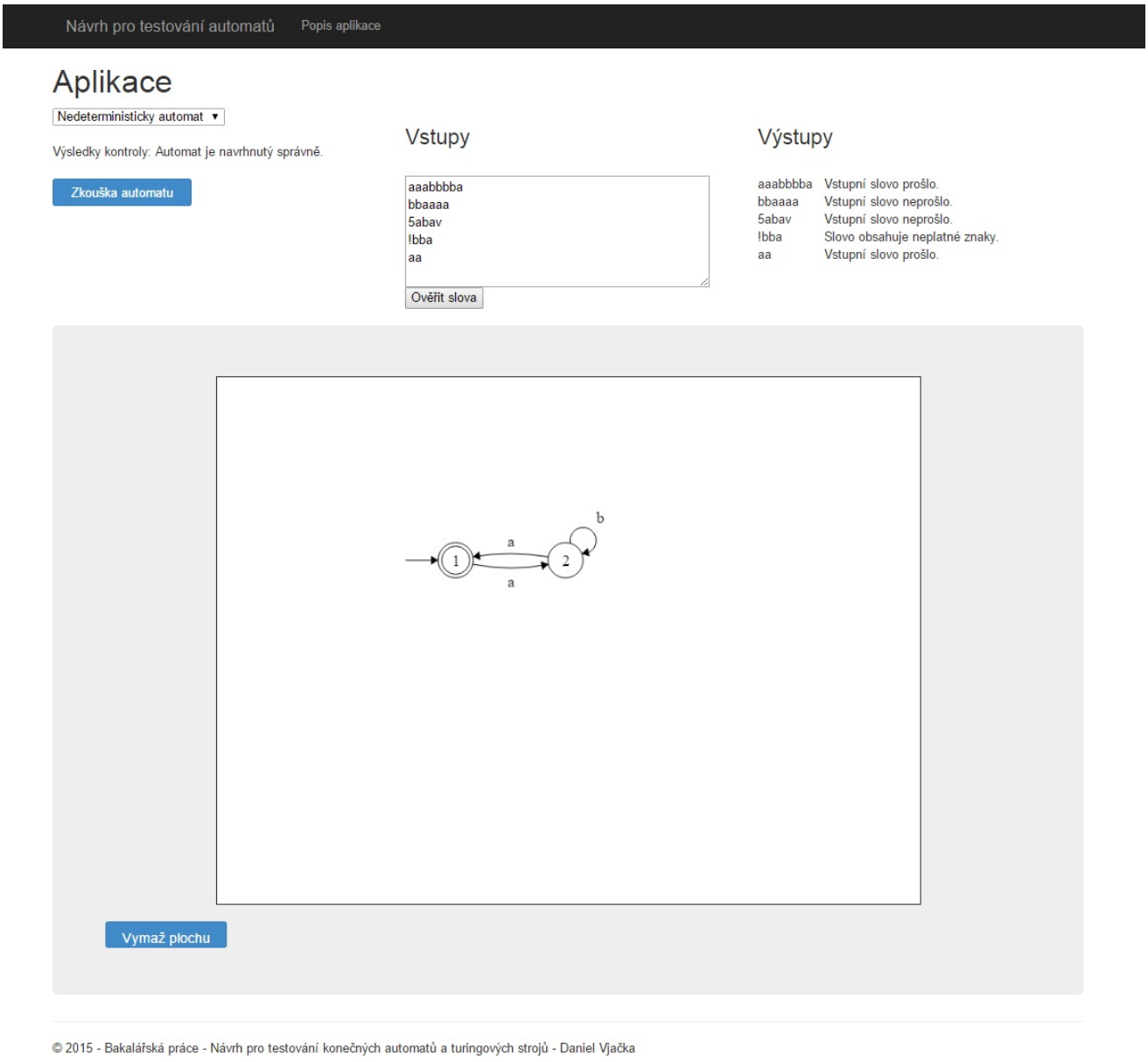


Obrázek 6: První návrh grafického prostředí

Canvas element bych se měl snažit udělat co největší. Uživatel neboli student bude mít pak větší kreslící plochu, což určitě ocení při složitějších automatech. Na stránku budeme muset zakomponovat uvedené prvky. V pozdější implementaci se budu snažit vměstnat uvedené prvky do úhledné webové aplikace.

6.2 Výsledné grafické prostředí

Po kompletním naprogramovaní aplikace jsme museli udělat grafické rozhraní aplikace. Oproti návrhu se trochu liší, jelikož se mi zvýšily požadavky a musel jsem přidat určité prvky. Horní část je rozdělena do tří pomyslných ovládacích sloupců. Toto rozdělení je vytvořeno pomocí mnou zmiňovaného frameworku Twitter Bootstrap. Použili jsme na to divy, kterým jsme přiřadili potřebnou class.



Obrázek 7: Výsledné grafické prostředí

První sloupec obsahuje ovládací prvky, které jsou stejné pro všechny tři výpočetní modely. Tento sloupec obsahuje DropDownList, který slouží k vybrání modelu a tím i dalších ovládacích prvků. Tuto funkcionalitu nám řeší javascript, kdy při změně v listu nastaví příslušné komponenty na skryté nebo viditelné. Dále zde máme výsledek kontroly, který je zde pro všechny modely a ověří zda je správný z matematického hlediska. Poté je zde umístěno tlačítko, které spouští příslušnou funkci pro ověření modelu.

Další druhé dva sloupce se mi mění v závislosti na zvolené volbě. U DKA a NKA máme prvek TextArea pro zadávání více možných vstupů pro automat, které musí být odděleny entrem. Poté zde máme tlačítko pro ověření vstupních slov automatem. Poslední sloupec je určen pro vypsání výstupů aplikace, které jsou u automatů řešeny pomocí tabulky. Tabulka je nutná, protože vstupů může být více a pro více vstupů musíme vypsat výstupy.

Poslední velkou část tvoří prvek zvaný jumbotron, opět z frameworku Twitter bootstrap do kterého jsem vložil canvas element a tlačítko pro vymazání canvasu neboli kreslící plochy. Toto tlačítko opět volá funkcionalitu z javascriptu, kdy nám to vymaže data z localStorage a celou kreslící plochu.

Jak jsme si řekli u volby Turingového stroje se nám mění druhý a třetí sloupec, což odpovídá vstupům a výstupům.

Obrázek 8: Návrh výstupů pro Turingové stroje

Druhý sloupec u Turingového stroje požaduje po uživateli zadání vstupního a výstupního řetězce. Po vložení vstupního řetězce do Turingového stroje provede program výpočet a ve třetím sloupci, což je sloupec pro výsledky vypíše uživateli výsledek. Ten se buď shoduje s uživatelovým nebo ne.

7 Aplikace

V této kapitole se už budu zabývat samotnou implementací a výslednou aplikací. Popíši zde první část, kde jsem programoval v javascriptu a byly vytvořeny dva javascriptové soubory, *fsmautomat* a *automatobjects*. Poté je vytvořena ve Visual Studiu webová aplikace asp.net a zde element canvas, na který se tyto dva javascriptové soubory odvolávají. Algoritmy pro řešení správnosti jsou programovány v jazyce C#. Proto zde popíšu druhou část a také jednotlivé algoritmy a postupy řešení. Hlavní logika je obsažena ve třídě *Automat*.

7.1 První část

V první je vytvořený designér, který je rozdělen do dvou souborů a to automatobjects a fsmautomat. V soboru automatobjects jsou už jak z názvů vyplývá objekty automatu jako například.

- Prechod
- Uzel
- SmyckaPrechod
- StartPrechod
- TempPrechod

Vysvětlíme si funkcionalitu těchto objektů a dále funkce v fsmautomat, které nám slouží ke zpracovávání příkazů.

7.1.1 automatobjects

Jako první se zaměříme na javascriptový soubor automatobjects. Je to soubor, kde máme umístěny všechny objekty, v krátkosti si uvedeme k čemu se používají. Tyto objekty jsou poté uloženy v poli v souboru fsmautomat. Zde se využívají k dalším funkcím.

Jako první objekt je zde Přechod, kterému se v atributech pošle uzel A a uzel B. Dále pomocí vlastnosti *prototype* si určíme, jaké vlastnosti má vytvořená instance mít. Tento objekt musí mít základní proměnné a to uzelA, uzelB, text a další proměnné, které používám v mnou vytvořených funkcích.

Pomocí vlastnosti protoype si vytvářím několik vlastností objektu Přechod. Mezi nejzákladnější patří funkce Prechod.prototype.kresli. Tato funkce je zde k vykreslení přechodu, kdy v sobě uchovává funkce na kreslení šipky, která je nadefinována v souboru fsmautomat. Dále se také zajímá o vykreslení textu, kdy k tomu opět využívá funkci napiš text z druhého souboru. O těchto funkcích si více řekneme v podkapitole, která se zajímá o tento soubor a funkce v něm obsažené. Mezi další hojně využívané funkce jsou funkce, které nám vracejí kotevní body daného přechodu.

Dále máme nadefinované také objekty, jako například SmyčkaPřechod. Což je přechod, který směřuje sám do sebe. Tato třída přijímá jako atributy uzel a proměnnou

mouse. Proměnná mouse se nám zasílá ze souboru fsmautomat a pro vysvětlení to je relativní pozice myši.

Opět zde máme pomocí vlastnosti prototype přidanou funkci kresli. Tato funkce nám v tomto případě vytvoří přechod typu smyčka, což znamená že vytvoří přechod sama do sebe. Tím pádem musíme zavolat na canvas funkci, která nám vykreslí kruh, což je funkce *arc()*. Poté nesmíme zapomenout na funkci ze souboru fsmautomat, která nám poslouží k napsání textu k přechodu. Nakonec se zavolá funkce na vykreslení šipky.

Poté je tu také start přechod, který se vytváří z prázdného prostoru do uzlu. Tímto přechodem se nám značí startovní uzel a start celého automatu. Opět má tento přechod základní vlastnosti jako předchozí a také má velmi jednoduchou implementaci funkce kresli, která je velmi podobná jako u objektu TempPřechod.

A jako poslední objekt z přechodů zde máme TempPřechod. Což je přechod, který je vytvořený pouze po dobu kdy táhneme myší. Na rozdíl od ostatních objektů je velmi jednoduchý a proto si zde uvedeme jeho celou implementaci, kdy se nejenom seznámíme s objektem TempPřechod, ale také s použitím vlastnosti prototype.

```
function TempPrechod(from, to) {
  this.from = from;
  this.to = to;
}
TempPrechod.prototype.kresli = function(c) {
  // Kresli link
  c.beginPath();
  c.moveTo(this.to.x, this.to.y);
  c.lineTo(this.from.x, this.from.y);
  c.stroke();
  // Kresli hlavicku sipky
  kresliSipka(c, this.to.x, this.to.y, Math.atan2(this.to.y - this.from.y, this.to.x - this.from.x
      ));
};
```

Výpis 10: Implementace objektu TempPrechod

Z kódu si můžeme vyčíst, že přijímá dva atributy a to odkud a kam až se má vytvořit. Poté je pomocí mnou zmiňované vlastnosti prototype vytvořená funkce kresli, která nám vykreslí čáru a na konci hlavičku šipky.

Nakonec si uvedeme objekt typu Uzel. Tento objekt je používán jak z názvů vyplývá pro vykreslování uzlů, neboli stavů automatu. Uzly mají také jednu důležitou proměnnou a to je proměnná typu bool zvaná přijímací stav. Do této proměnné se nám ukládá, zda je daný stav přijímací či nikoliv. Implementací je tento objekt velmi jednoduchý. Jako u každého objektu si uvedeme jak se provádí vykreslování. U funkce kresli se nám vykreslí kruh pomocí funkce *arc()*, zavolá se funkce napiš text pro vypsání textu a poté, pokud je daný stav přijímací tak vykreslí ještě jedno kolečko, které nám určuje, že stav je přijímací. Velikost kolečka se nám určuje pomocí globální proměnné uzelRadius.

Další funkce, které tento objekt obsahuje jsou funkce, které se využívají v souboru fsmautomat při pohybu myši a pohyby s uzly. V této bakalářské práci vysvětluji jenom zlomek použitých funkcí v celém programu.

7.1.2 fsmautomat

Nyní se dostáváme k souboru fsmautomat, kde zachytáváme události, jako například pokud nám uživatel klikne do obrazovky. V tomto souboru také převádíme náš automat do textového podoby a následně ho posíláme do druhé části, kde řešíme algoritmické vyjádření.

Ze začátku máme nadefinovaných pár proměnných, z těch zajímavých například uzly[] a prechody[]. Ty uchovávají informace o všech uzlech a přechodech a můžeme s těmito informacemi manipulovat. Právě při tvoření textového formátu využíváme tyto pole k procházení jednotlivými prvky. V kódu funkce getString si můžeme povšimnout vkládání uzlů do výstupního řetězce. Nejprve si nadefinujeme příkaz for, který nám prochází všechny uzly a následně si je přiřadí do proměnné uzel, z které taháme proměnné text a přijímacíStav, který je typu true nebo false. U tvoření výstupního formátu z přechodů máme zápis trochu složitější, jelikož si rozlišujeme, zda je stav typu Smyčka,Start nebo obyčejný. A následně stejným způsobem přidáváme přechody do našeho výstupního řetězce.

```
function getString() {
    var VysRetezec = "[";
    for (var i = 0; i < uzly.length; i++) {
        var uzel = uzly[i];
        VysRetezec += "{";
        VysRetezec += "(namenode:" + uzel.text + ")";
        VysRetezec += "(konecnode:" + uzel.prijimaciStav + ")";
        VysRetezec += "}";
    } // .. kod – vlozeni prechodu do vystupniho retezce
```

Výpis 11: Funkce na tvoření textového formátu

Následně si v javascriptu nastavíme, že po kliknutí tlačítka se nám zavolá tato funkce a pomocí funkce __doPostBack si zašleme náš argument, neboli výstupní řetězec do našeho C# kódu.

V kódu nalezneme mnoho funkcí, ale uvedu pouze pár pro představu jak jsem pracoval s kódem a jak jsou naprogramovány určité prvky. Uvedeme si funkcí, která nám určuje co se stane pokud myší provedeme tzv. dvojklik.

Tato funkce si vezme pozici myši a zavolá funkci zakazZkouska(), která nám zakáže tlačítko, aby uživatel nemohl zmáčknout tlačítko pro ověření vstupního řetězce dřív než bude automat vyzkoušen a z matematického hlediska bude dávat smysl. Poté se nám uloží do proměnné zvolenyObjekt z pozice myši. Na tuto proměnnou se budeme ptát, zda je prázdná nebo je zde instance Uzlu. Pokud bude prázdná, tak se nám zde vytvoří uzel nový. Tento nový uzel bude přidán do naší proměnné uzly[], o které jsme mluvili na začátku této podkapitoly. Pokud zde bude už vytvořená instance, tak se nám změní stav uzlu. V případě normálního uzlu na uzel konečný.

Poté se v tomto souboru nacházejí funkce, na které jsme se odkazovali v minulé podkapitole. Jedna z nich je například funkce na vykreslování šipky. Další z důležitých funkcí je funkce, která slouží k uchovávání objektů tzv. Backup funkce. Objekty se ukládají do lokálně vytvořeného pole pro uzly a přechody které se následně uloží do naší localSto-

rage určené pro automaty. Tato funkce se nám vždy při zavolání funkci kresli(). Poté je zde také funkce na obnovení naší zálohy a tato funkce se zavolá vždy když je náš dokument "ready".

Dále jsou zde také funkce pro stisknutí klávesy shift, pomocí které můžeme následně kreslit přechody. Tento soubor obsahuje i funkci pro kreslení textu. Tímto jsme si shrnuli co tento program obsahuje a přesuneme se k části druhé.

7.2 Druhá část

V implementaci druhé části si popíši třídu Uzel a třídu Automat. Dále si vysvětlíme průchod algoritmem automatů a Turingových strojů.

7.2.1 Třída Uzel

Tato třída tvoří jednu z hlavních částí programu. V této třídě jsou uloženy všechny hlavní informace, které algoritmy budou potřebovat průchodem automatem. Pro každý jednotlivý uzel je zde vytvořená samostatná instance. Tato instance si z textového formátu vytáhne základní informace a to *text*, který je využíván pro název uzlu. Poté dvě proměnné typu bool pro startovní a konečné prvky. Jako poslední proměnnou je zde Dictionary<Key,Value>. V této proměnné jsou uložené přechody na další uzly. Dictionary je speciální list, který má ke každé hodnotě přiřazený klíč. Zde je jako klíč uvedený proměnná typu string, což je název přechodu. Jako hodnota je zde proměnná typu List<Uzel> což je z toho důvodu, že u nedeterministických automatů může nastat případ, kdy pro jeden přechod můžou být dva uzly. U dictionary to nemohu řešit jinak, jelikož nepodporuje duplicitu klíčů. Pokud bychom nechtěli volit tuto verzi je možné použít kolekci Lookup. Já osobně jsem z Dictionary pracoval a usoudil jsem, že toto bude lepší varianta tak s tímto pracuji takto.

```
public class Uzel
{
    //Promenne
    public string  text;
    public bool  start;
    public bool konec;
    public Dictionary<string,  List<Uzel>> prechody;
    .
    .
}
```

Výpis 12: Proměnné třídy Uzel

Dále se v této třídě nachází funkce PridejPrechod, která si bere proměnné - text,Uzel a typautomatu. Zde přidáváme přechody k jednotlivým uzlům. Zasíláme si zde text z přechodu, ale musíme si dát pozor na to, že přechod může být proveden pro více znaků vstupní abecedy. Pokud text obsahuje čárku, tak se provede útržek kódu níže. Text si rozdělí do pole stringů, který následně prochází každý znak. Další krok je, že zjišťuje zda v daném uzlu je v Dictionary přechodů s uvedeným klíčem nějaká hodnota. Pokud

ano, tak jednoduše přidá již k existujícímu listu hodnot nový uzel. Pokud neobsahuje, tak vytvoří nový list uzlů a přidá ho do kolekce přechodů.

```
// utrzek kodu
string [] splittexty = text.Split(',');
foreach (string splittext in splittexty)
      {
        if (prechody.ContainsKey(splittext))
        {
            prechody[splittext].Add(uz);
        }
        else
        {
             List<Uzel> uzly = new List<Uzel>();
             uzly.Add(uz);
             prechody.Add(splittext, uzly);
        }
       }
```

Výpis 13: Funkce PridejPrechod

Dále tato třída obsahuje pouze konstruktory, které si myslím jsou jasné.

7.2.2 Třída Automat

Jako další se vrhneme na třídu Automat. V této třídě provádíme veškerou funkcionalitu a také zde jsou všechny algoritmy pro řešení automatů a Turingového stroje. Funkce které jsou v této třídě jsou:

- konstruktor Automat
- KontrolaAutomatu
- Over
- AlgDKA
- AlgNKA
- NKA
- ALgTNG

Některé si vysvětlíme pouze v krátkosti. A to například, jako funkce KontrolaAutomatu, která pouze kontroluje automat zda je správně navrhnut. Více k podmínkám a to co jsme museli kontrolovat v podkapitole Podmínky a výstupy aplikace. Další jednoduchá funkce je funkce Oveř. Tato funkce podle zvoleného typu zavolá příslušnou třídu, která obsahuje algoritmus a vrátí booleovskou hodnotu true nebo false souběžně s příslušným výstupem. První složitější funkcí, kterou si lehce probereme je konstruktor, který si bere tři string atributy. První, který předává data pomocí námi navrženého textového formátu,

druhý, nám posílá typ automatu a poslední je proměnná *string out vystup*. Poslední proměnná výstup a její modifikátor *out* slouží k vrácení výstupu aplikace. Hned ze začátku si opět vytvoříme Dictionary<string,Uzel>, jako klíč je zde uveden název uzlu a hodnota je zde instance třídy Uzel. Tato kolekce bude sloužit k uchovávání všech uzlů.

Jako další krok v této kolekci je rozparsování našeho textového formátu do pole stringů. Nyní máme vytvořeny dvě pole stringů, jeden pro uzly a druhý pro přechody. Pomocí příkazu foreach si projedeme jednotlivé uzly a přidáme je do naší kolekce.

Podobný krok je pro přechody. Opět procházíme jednotlivé přechody a přiřazujeme je ke správným uzlům. Všechny tyto informace, jako například z kterého do kterého uzlu je vytvořen přechod, jsou obsaženy v textovém formátu. V procházení si také ukládáme do lokálních proměnných důležité informace, které dále využijeme ve funkci KontrolaAutomatu. Zde budeme například potřebovat vědět počet startovních prvků, počet konečných prvků atd. Jediný rozdíl od přidávání uzlů do kolekce je v tom, že pro přechody máme vytvořenou vlastní funkci PridejPrechod, které pošleme text, daný uzel do kterého je přechod a typ automatu.

Na konec zavoláme funkci KontrolAutomatu, která nám vše zkontroluje zda je automat správně navrhnut.

```
public Automat(string data, string typautomatu, out string vystup)
{
    uzlydic = new Dictionary<string, Uzel>();
    .. // kod – nize prechod mezi uzly
    foreach (var uzel in uzly)
      {
        var members = uzel.Split(new[] { '(', ')' }, StringSplitOptions.RemoveEmptyEntries).
              Select(s => s.Split(new[] { ':' })) ;
        node = new Uzel();
             // Prechod mezi polozky v uzlech
           foreach (var member in members)
               {
                 if (member[0] == "namenode")
                   node.text = member[1];
                   ... //kod
                   uzlydic.Add(node.text, node);
                   ... //kod
                } .. //kod
       }
     foreach (var prechod in prechody)
        {... //kod
        foreach (var clen in clenove)
             {.. //kod
              if (clen[0] == "nodeA")
                 nodeA = uzlydic[clen[1]];
            }
               if (typ == "Link")
                 nodeA.PridejPrechod(text, nodeB, typautomatu);
        }
}
```

Výpis 14: Konstruktor třídy Automat

Následně si probereme tu nejdůležitější část a to jsou algoritmy které řeší příslušné výpočetní moduly.

7.2.3 Algoritmus pro DKA

Nejjednodušší algoritmus zde uvedu ten pro deterministické automaty. Jelikož mám uzly řešeny pomocí kolekcí a stejně mám řešeny také přechody, proto je přechod mezi uzly vcelku jednoduchý.

Funkce je typu bool, kdy vrací true nebo false v závislosti na průchodu algoritmem. Této funkcí předáváme potřebný vstup, který bude procházet a zjišťovat zda je automat platný.

Nejprve si zjistíme, který uzel je počáteční a přiřadíme ho do proměnné uzel. Poté budeme procházet znak po znaku ze vstupu a pro jednotlivé znaky zjišťovat, zda má daný uzel přechod. Toto zjišťujeme pomocí odkázání se na Value v Dictionary prechody. Tento postup provádíme do doby, dokud nedojdeme do posledního možného uzlu.

Pokud jsme algoritmem došli do posledního možného stavu, který je zároveň konečný, tak se nám vrátí returnem uzel.konec, čímž se nám vrátí true a víme, že vstup prošel daným konečným automatem.

Je zde také speciální proměnná znFound, která hlídá zda byl daný znak nalezen. V přiloženém kódu si můžeme povšimnout, že se proměnná znFound nastaví na true pouze v případě, že se nalezne přechod s daným znakem. Na konci tuto proměnnou budeme muset znova nastavit na false. Pokud by to znak nenašlo, v proměnné zůstává false a celá funkce nám returnem vrátí false, což značí že vstup neprošel automatem.

```
public bool AlgDKA(string vstup)
{
    Uzel uzel = null ;
      // .. zde umisten kod pro nalezeni startovniho uzlu
    bool znFound = false;
    foreach(char zn in vstup)
     {
        string znak = zn.ToString();
        foreach(var prechod in uzel.prechody)
          {
              if (prechod.Key == znak)
               {
                uzel = prechod.Value[0];
                znFound = true;
                break;
               }
          }
         if (znFound)
            znFound = false;
         else return false;
      }
     return uzel.konec;
}
```

Výpis 15: Algoritmus pro řešení deterministických automatů

7.2.4 Algoritmus pro NKA

Jako další si uvedeme algoritmus pro nedeterministické automaty. K vyřešení tohoto algoritmu jsem využíval dvou funkcí. AlgNKA a NKA. Kdy AlgNKA funkce je volána z funkce Ověř a zde se opět jako u deterministických automatů hledají startovní neboli počáteční prvky. U tohoto hledání ale nastává mírná změna, jak jsme si v teorii zmínili tak nedeterministický automat může obsahovat jeden a více počátečních stavů. Z tohoto důvodu jsme si pro počáteční stavy vytvořili proměnnou typu List<Uzel> a poté pro každý ze startovních prvků budeme volat funkci NKA.

U funkce NKA jsem využíval rekurzivního volání funkce. Tato funkce přijímá v atributech string vstup a proměnnou typu Uzel. Zde jako první zjišťuji, zda vůbec daný uzel má přechod pro nadcházející písmeno ze vstupního řetězce. Pokud nemá, tak je zbytečné pokračovat ve funkci a ihned vracíme false. Zapoměl jsem zmínit, že tato funkce je opět typu bool. Kdy se mi ve funkci AlgNKA ukládá výsledek daného algoritmu a vrátí výsledek na obrazovku.

Pokud jsme zjistili, že v přechodech se daný znak vyskytuje tak jak vidíme v kódu, zjišťujeme kolikrát se daný znak vyskytuje. Pokud se nám daný znak vyskytuje pouze jednou, tak si rekurzivně zavoláme funkci NKA a předáme vstup bez prvního znaku a také uzel, který se nám vyskytuje v kolekci přechody s klíčem znak.

Poté také můžeme ale zjistit, že nedeterministické automaty můžou mít pro stejný znak v přechodech List více uzlů. Proto si pro každý uzel z listu rekurzivně zavoláme tuto funkci a opět ji předáme vstup bez prvního znaku a daný uzel. Poté vrátíme naší lokální proměnnou vys, čímž zjistíme zda vstupní řetězec prošel automatem či ne.

```
// ..  kod
if (zacuzel.prechody[znak].Count==1)
    vys=vys||NKA(vstup.Substring(1),zacuzel.prechody[znak][0]);
if  (zacuzel.prechody[znak].Count >= 2)
  {
   foreach (var uzel in zacuzel.prechody[znak])
       {
         vys=vys||NKA(vstup.Substring(1), uzel);
       }
   }
return vys;
```

Výpis 16: Algoritmus pro řešení nedeterministických automatů

7.2.5 Algoritmus pro TNG

Jako poslední zde máme algoritmus pro řešení Turingových strojů. Zde nemůžeme pouze jednoduše procházet uzly, jelikož Turingův stroj má taky funkci zapisování a také posouvání hlavy po pásce vlevo nebo vpravo. Další změna v tomto algoritmu je, že v textu pro přechody nejsou pouze znaky pro posun přechodu ale veškeré instrukce pro provedení kódu. Na tento fakt bychom se měli zaměřit a vzít v potaz to, že budeme muset v algoritmu rozparsovávat tento text a s danýma instrukcema dále pracovat. U Turingového

stroje je ještě jedna vcelku zásadní vyjímka a to ta, že má jako atribut výstup, který nám dokáže na obrazovku vypsat výsledný řetězec.

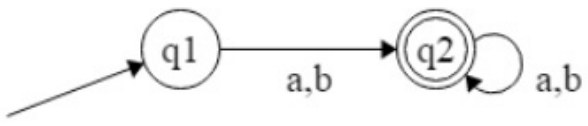
Začátkem algoritmus připomíná algoritmus DKA, jelikož si opět najdeme startovní prvek. Poté za pomocí příkazu while procházíme jednotlivé uzly za podmínky *!uzel.konec*. Kde si do proměnné znak typu char ukládáme vstupní znak, který se nachází na námi potřebné pozici. Poté si pomocí foreach pro daný uzel procházíme jednotlivé přechody v uzlech. V této smyčce musíme námi daný text rozparsovat do třech potřebných informací. Textový formát, který nám tam přijde, bude vypadat takto - 0/X/R - kdy první informace znamená co v tomto přechodu stroj musí přečíst. Druhá informace znamená, co stroj na danou pozici zapíše a poslední informace znamená posun hlavy v našem případě doprava. Pokud přechod obsahuje znak se vstupu, poté se vykonají potřebné operace. A to, jak jsem říkal, se nám zapíše na danou pozici pomocí příkazu *vstupy[pozice] = zapis;*. Pak se posune hlava po pásce vlevo či vpravo. Počítač, ale žádnou hlavu neobsahuje, takže se pouze změní hodnota proměnné pozice v závislosti na směru posunu.

U posunu si ale musíme dávat pozor, abychom nevystoupili mimo nadefinované pole. Proto pokud se pohybujeme doprava, musíme zjistit velikost pole a pokud je velikost menší než náš posun, tak zde musíme pro tyto políčka naalokovat místo v paměti. To samé musíme zjišťovat při posunu vlevo. Při posunu vlevo, pokud je pozice na vstupním řetězci rovna 0, posuneme celý řetězec o jedno pole doprava a naalokujeme si první pole prázdnou hodnotou, na kterou případně můžeme zapisovat.

Tento algoritmus se nám opakuje, jak už jsme si řekli do doby, dokud nenarazíme na konečný uzel. Zpět vracíme náš výstup, na který jsme zapisovali a po kterém jsme se pohybovali a vypíšeme ho uživateli na obrazovku.

7.3 Textový formát

Jeden z požadavků u mé bakalářské práce, jak už jsem mnohokrát zmínil bylo vytvoření textového formátu s určitým ohledem na čitelnost člověkem. Na obrázku si povšimněme, že jsme si nadefinovali jednoduchý dvou stavový deterministický automat a jeden z těchto stavů jsme si označili jako koncový. Poté zde máme nadefinovány 3 stavy, jeden je Start link, který míří do stavu q1. Dále zde máme přechod mezi dvěma uzly, který jde ze stavu q1 do stavu q2 pomocí znaků a nebo b. A poslední je přechod typu smyčka, který jde sám do sebe. Zde zůstává při načtení jakéhokoliv písmene ze vstupní abecedy.



Obrázek 9: Automat, který následně převedeme na text

Po stavovém diagramu si převedeme automat do textového formátu. Formát je vytvořen javascriptem a poté poslán a rozparsován v C#. Pro jednoduchou práci jsem si vytvořil jednoduchou a jasnou strukturu formátu, která je následující. Hranaté závorky

[] nám uzavírají všechny uzly a poté všechny přechody. Složené závorky {}, nám uzavírají jednotlivé uzly a přechody. A nakonec zde máme kulaté závorky (), které nám uzavírají jednotlivé informace.

U uzlů si předáváme pouze název uzlu a zda je uzel konečný. Při přechodech si předáváme typ přechodu, uzly odkud a kam, abychom mohli určit směr přechodu a nakonec text daného přechodu.

[{(namenode:q1)(konecnode:false)}{(namenode:q2)(konecnode:true)}]
[{(nodeA:q1)(type:Link)(nodeB:q2)(text:a,b)}{(nodeA:q1)(type:StartLink)(text:)}
{(nodeA:q2)(type:SelfLink)(text:a,b)}]

7.4 Podmínky a výstupy aplikace

Pokud programujeme nějakou aplikaci pro uživatele, tak nesmíme zapomenout na to, že člověk není dokonalý. Uživatel může zadat například špatný vstup a celá aplikace se nám může zhroutit kvůli jednomu špatnému znaku. Tato část tedy s určitou úrovní nám zjišťuje, zda je automat navrhnut matematicky správně či nikoliv. Zde si vypíšeme, na co nesmíme zapomenout, jak jsme to ošetřili a jakými výstupy to dáme uživateli vědět.

Výstupy, které jsou stejné pro všechny tři výpočetní matematické modely jsou následující.

- Název uzlu je stejný, pro více než jeden uzel! Musíš pozměnit jména uzlu.
- Některý z uzlů nemá jméno nebo je zadaný neplatný název. Pro název použij pouze malou, velkou abecedu nebo čísla.
- Některý z přechodů nemá hodnotu nebo je zadaný neplatný název. Pro název použij pouze malou, velkou abecedu nebo čísla.
- Automat je navrnutý správně.

Jako první kontrola nám probíhá kontrola týkající se názvu uzlu. Samozřejmostí je, že automat nemůže v žádném případě obsahovat uzly, které budou mít stejné jméno pro více než jeden uzel. Poté by mohla v průběhu automatu nastat situace, že automat nebude vědět, v kterém je uzlu a kterým směrem se vydat. Toto se nám řeší v konstruktoru Automat a přesněji při přidávání uzlu do Dictionary si odchytávám vyjímku a pokud jsou uzly stejné tak vyhazuji chybovou hlášku.

Další z ověřování nastává v mé funkci KontrolaAutomatu. V této chvíli máme vytvořenou Dictionary uzlů a ty jsou naplněny svými přechody. Budeme si procházet pomocí funkce foreach jednotlivé uzly a poté pro každý klíč uzlu z Dictionary si ověříme, zda není prázdný. Také si ve stejné chvíli pomocí regulárních výrazů budeme kontrolovat, zda klíč uzlu neboli název uzlu je složen pouze z povolených znaků. Naše povolené znaky jsou - malá abeceda, velká abeceda nebo číslice. Kontrola povolených znaků, jak jsem řekl je pomocí regulárních výrazů, což je speciální řetězec znaků, který nám ověří, zda náš vstup splňuje naše požadována pravidla 11 . A to, že jsou to námi povolené znaky. K tomuto nám pomůže třída *Regex*, která slouží ke zkoušení regulárních výrazů. Tato třída má metodu *isMatch()*, která si bere jako argumenty náš vstup, což bude náš

klíč uzlu a poté náš regulární výraz samotný. Regulární výraz pro ověřování vypadá takto Regex.IsMatch(uzel.Key, @"^[a-zA-Z0-9]+$"]) 3. Tato metoda je typu bool a proto nám vrátí false nebo true podle toho, zda to prošlo nebo ne. Podle toho také můžeme uživateli vypsat výstup, jestli je název platný nebo v něm má chybu a bude si ho muset opravit.

Podobný postup jako u uzlů provádíme u přechodů. Pokud se nacházíme v aktuálním uzlu, tak poté pro každý z přechodů provedeme stejnou kontrolu jako u uzlu. Opět zkontrolujeme jestli není prázdný a zda obsahuje nějaké neplatné znaky. Po kontrole vyhodíme vhodný výsledek operace.

Jako poslední zde máme oznámení, že je automat správně navrhnut. Tuto hlášku si přidáme do proměnné *vystup* typu string ihned na začátku funkce KontrolaAutomatu a pokud náhodou nastane nějaká chyba, tak se do vstupu přiřadí příslušná hláška a vrátí false. Dále si uvedeme výstupy, které nám můžou nastat, pokud se zaměříme na daný výpočetní model, což může být deterministický automat, nedeterministický automat nebo Turingův stroj. U detereministického automatu to může být.

- Neplatný automat, počet konečných prvků je nulový
- Neplatný automat, počet startovních prvků je nulový nebo větší než jedna
- Neplatný automat, pro každý uzel není stanovena přesně jedna cesta

Máme zde tři možnosti výstupu pro uživatele. Ohledně startovních a konečných prvků je to jednoduché. Máme vytvořené dvě proměnné, do kterých si ukládáme počet začátečních stavů a počet konečných stavů. Pokud nalezneme nějaký přechod, který bude typu StartPřechod což znamená, že uzel na který tento přechod ukazuje je začáteční. Do proměnné pocetstart si tedy přičteme 1. To samé děláme s konečnými prvky. Informaci o konečném prvku si vytáhneme s textového formátu, kdy uzel v sobě má uloženou informaci, jestli je konečný či nikoliv. Pokud nalezneme stav, který je konečný, tak přičteme jedničku. Z teorie již víme, že DKA můžou mít pouze jeden startovní prvek a také musí mít minimálně jeden konečný prvek, aby automat mohl skončit. Pokud se náhodou nebudeme držet teorie, uživateli opět vypíšeme, že se nedrží matematických definicí a musí to opravit.

U deterministických automatů máme ještě jednu podmínku, na kterou nesmíme zapomenout a to, že pro každý uzel musí být stanovena přesně jedna cesta. Toto ověřujeme následujícím postupem.

V inicializaci přechodů pro každý uzel máme sestavený HashSet stringů, který nám slouží k uchovávání vstupní abecedy. Při přidávání textu do přechodů si kontrolujeme, zda se nám následující znak nachází ve vstupní abecedě, pokud se ve vstupní abecedě nenachází, následně si jej přidáme. Vstupní abeceda je tvořena ze všech znaků, které se nacházejí kdekoliv v celém automatu. Níže uvedený útržek kódu se nachází ve funkci KotrolaAutomatu, jako první krok je, že si procházíme všechny uzly. U každého uzlu si kontrolujeme zda Dictionary přechody má stejnou velikost, jako velikost našeho HashSetu abecedy. Pokud je velikost větší, tak to znamená, že pro každý uzel není zvolená přesně jedna daná cesta. Vnořený příkaz if nám řeší, jestli pro znak ze vstupní abecedy

je zvolen pouze jeden přechod, jelikož se nacházíme v deterministickém konečném automatu.

```
// .. kod – prochazeni uzlu
if (uz.Value.prechody.Count == abeceda.Count)
{
    foreach (var pre in uz.Value.prechody)
        {
            if (pre.Value.Count != 1) {
                vystup = "Neplatny automat, pro kazdy uzel neni stanovena presne jedna
                    cesta.";
                return false; }
        }
}
else {
  vystup = "Neplatny automat, pro kazdy uzel neni stanovena presne jedna cesta.";
  return false; }
```

Výpis 17: Kontrola cest v deterministickém konečném automatu

Dále se zaměříme na nedeterministický automat, u kterého řešíme pouze dvě podmínky, které se podobají DKA.

- Neplatný automat, počet konečných prvků je nulový
- Neplatný automat, počet startovních prvků je nulový

U nedeterministického konečného automatu z definice víme, že počet konečných a startovních stavů může být libovolný počet pouze ne nulový. Toto bylo řešeno stejně jako u deterministického automatu. Znovu jsme si do lokálních proměnných ukládali kolikrát se nám nachází v celém automatu startovních a konečných prvků a pouze pomocí příkazu if ověřili, zda nejsou nulové. Pokud by jeden z nich byl, tak to uživateli opět vyhodí upomínku na úpravu svých chyb.

Taky zde nemusíme řešit přechody pro dané uzly. Jelikož pro znak z abecedy může být vytvořeno libovolně mnoho cest, nebo také cesta žádná. Toto nám zjednodušilo podmínky u NKA, ale algoritmus, který řeší nedeterministický automat je složitější.

A jako poslední zde máme Turingův stroj. Znovu stejně jako u automatů musíme ověřovat počty startovních a konečných prvků. Poté se nám vypíše hláška ohledně konečných prvků jako v minulých případech. Změna zde nastává v případě ověřování vstupního textu u přechodů. Tento musí být pro náš výpočet specifický. Nyní si uvedeme možné výstupy při ověřování.

- Neplatný stroj, text přechodu neobsahuje dvě lomítka
- Neplatný stroj, za posledním lomítkem musí být R,L, nebo S
- Neplatný stroj, text přechodů obsahuje neplatné znaky
- Neplatný stroj, za posledním lomítkem musí být pouze jeden znak

- Neplatný stroj, mezi lomítky musí být pouze jeden znak
- Neplatný stroj, neobsahuje první znak pro přečtení

Můžeme si povšimnout, že program u tohoto kontroluje text u přechodů a může zde nastat velké množství chyb. Uživatel by se mohl jednoduše splést a program tímto shodit. Formát textu přechodu vypadá následovně *a/b/R*, který má následující funkce čti/zapiš/-posuň se. Tímto přechodem řekneme stroji přečti "a"zapiš znak "b"a posuň se doprava , jelikož R-Right je znak pro posun doprava.

Tímto chápeme základní formát vstupu a vidíme, že musíme ověřovat každý přechod v každém uzlu. Nejprve ověřujeme zda přechod obsahuje dvě lomítka. Dále si ověřujeme, jestli je mezi lomítky a za lomítky nějaký text, bez textu bychom neměli potřebné informace pro vykonání přechodové funkce Turingového stroje. Další ověření přichází v počtu znaků u lomítek. Stroji musíme dávat přesné instrukce a proto pro funkci zapiš a posuň se musíme zasílat pouze jeden znak. Tímto stroj bude vědět, co má zapsat a kam se posunout. A jako poslední je zde kontrola platných znaků v přechodové funkci. Toto ověřování opět probíhá stejně jako jsme si již uvedli, pomocí regulárních výrazů.

8 Závěr

Cílem mé bakalářské práce bylo vytvořit nástroj pro návrh a testování konečných automatů a Turingových strojů. Výsledná práce splnila požadavky a je rozdělena na dvě samostatné, propojitelné části. První část byla vytvořena v jazyce JavaScript a na druhou část a logiku automatu jsem si vybral jazyk C#, který jsme se učili v předmětu Programovací jazyky II. V tomto předmětu jsem pochopil základy a získal mírně pokročilou znalost tohoto jazyka a také implementaci a vývoj aplikací v prostředí .NET frameworku. Touto bakalářskou prací jsem nabyl větších zkušeností jak v jazyce JavaScript, tak také C#. Vzhledem k tomu, že tato bakalářská práce musela řešit také logiku Turingova stroje, tak jsem si musel nastudovat tuto látku. Tuto problematiku jsem neprobíral při studiu na vysoké škole a samostudiem jsem si zjistil všechny potřebné informace. Díky této práci jsem pochopil problematiku Turingových strojů, která je pro mne opravdu velmi zajímavá.

Celý nástroj je plně funkční a je ošetřen o možnost jakékoliv chyby uživatelem. Práce je vytvořena jako webová aplikace, proto jsem k tomu využíval webových prohlížečů. Musel jsem dbát na to, aby celá aplikace byla funkční v prohlížečích Firefox a Chrome k datu odevzdání mé práce.

Tento nástroj je možné do budoucna rozšířit. Jako velmi zajímavé a taky nápomocné k pochopení látky, by byla animace přechodů. Graficky by se mohlo znázornit, kde se uživatel v danou chvíli vyskytuje a který znak je na řadě. Tímto by uživatel měl možnost si odkrokovat každý znak se vstupního slova. Dále mě napadla možnost implementace zadání slovního vstupu a následné vygenerování automatu. V tomto případě si nejsem jist, zda by tento nástroj poté nebyl zneužit studenty při zkouškách a musela by se zvážit implementace této funkce. Osobně si myslím, že možností ke zlepšení je více. Téma ohledně konečných automatů a Turingových strojů skrývá hodně možností a velkou škálu funkcionality.

9 Reference

[1] Jančar, Petr, *Úvod do teoretické informatiky – učební text*, Ediční středisko VŠB-TUO, 2007.

[2] Sipser, Michael, *Introduction to the Theory of Computation*, Thomson Course Technology, 2006, ISBN 0-534-95097-3

[3] MSDN Microsoft Developer Network [online] [cit. 2015-03-20] Dostupný z WWW: <https://msdn.microsoft.com>.

[4] Hrančík M., Animace k předmětům teoretické informatiky [online] [cit. 2015-04-10] Dostupný z WWW: <http://www.cs.vsb.cz/kot/animace.php>

[5] Třídy složitosti a Turingovy stroje [online][cit. 2015-04-24] Dostupný z WWW: <http://www.algoritmy.net/article/5774/Tridy-slozitosti>

[6] Konečné automaty [online][cit. 2015-04-10] Dostupný z WWW: <http://www.matematika.cz/konecny-automat>

[7] ASP.NET framework [online][cit. 2015-04-15] Dostupný z WWW: <http://www.asp.net/web-forms/overview>

[8] Teorie konečných automatů [online][cit. 2015-04-10] Dostupný z WWW: <http://iris.uhk.cz/tein/teorie/konecnyAutomat.html>

[9] Twitter Bootstrap [online][cit. 2015-04-14] Dostupný z WWW: <http://getbootstrap.com/>

[10] ASP.NET WebForms [online][cit. 2015-04-18] Dostupný z WWW: <http://www.w3schools.com/aspnet/aspnet_intro.asp>

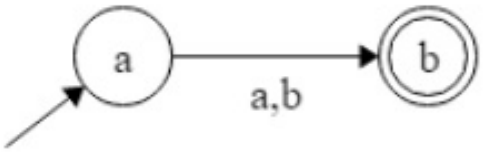
[11] Regulární výrazy [online][cit. 2015-04-25] Dostupný z WWW: <http://www.regularnivyrazy.info>

A Testování aplikace

Nakonec celého programovacího cyklu si musíme otestovat, jestli vše funguje jak má a zhodnotit výsledky aplikace. Momentálně jsme si v bakalářské práci vysvětlili první část, druhou část a také výstupy aplikace. Nyní si názorně vyzkoušíme jeden chybný příklad a druhý správný. Dále si zadáme, jaké jsou možné vstupy a výstupy.

A.1 DKA

Prvně začneme s deterministickým konečným automatem. Začneme se špatným příkladem, kde bude zadán špatně automat a ověříme zda funguje kontrola automatu a všech matematických náležitostí.



Obrázek 10: DKA - Test 1: Špatně zadaný automat

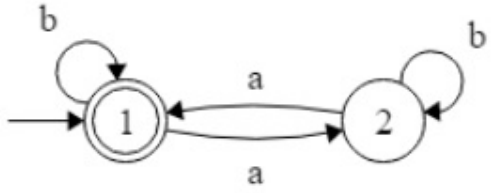
V našem návrháři jsme si vytvořili deterministický automat, který ale nemá pro každý znak se vstupní abecedy zadanou cestu a proto nám po kliknutí na tlačítko aplikace vyhodí, že máme špatně zadaný automat.

Obrázek 11: DKA - Test 1: Výsledek

Na obrázku si můžeme všimnout, že výsledek kontroly nám vyhodil chybovou hlášku. Kontroly automatu a výstupy aplikace jsme si probírali v minulé kapitole.

Dále si zadáme správný automat, který bude splňovat všechny matematické náležitosti a ověříme si možnost vstupů a výstupů aplikace.

Zadání automatu je vytvořit takový automat, který bude přijímat slova pouze se sudým počtem znaku "a". Proto, si v našem návrháři navrhneme automat a zadáme několik vstupů, které budeme chtít ověřit.



Obrázek 12: DKA - Test 2: Správně zadaný automat

Zde si můžeme povšimnout, že automat je správně navrhnut. Splňuje všechny matematické náležitosti a po kontrole automatu zadáme potřebné vstupy.

Obrázek 13: DKA - Test 2: Výsledek

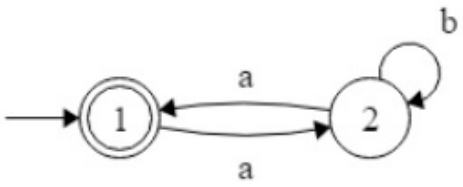
Pro zkoušku a ukázku jsme si uvedli více vstupů, které projdou a které neprojdou. Projdou pouze slova, která obsahují sudý počet znaků "a". Poté jsme si pro ukázku uvedli slova, která neprojdou. Neprojdou nám automatem slova, která obsahují lichý počet znaků "a". Poté nemůžou projít slova která sice obsahují sudý počet znaků "a", ale obsahuje nám číslice které nejsou ve vstupní abecedě. Další případ kdy vidíme, že nám automat vyhodí chybovou hlášku, je při načtení neboli vložení nepovoleného znaku. Teď jsme si uvedli všechny případy, které nám můžou nastat.

A.2 NKA

Pro vyzkoušení nedeterministického automatu si zadáme podobný případ jako v DKA. Pouze odebereme u prvního stavu přechod do sebe sama pomocí znaku "b", což je vlastnost tohoto automatu.

Poté si vyzkoušíme, zda automat obsahuje všechny matematické náležitosti a aplikace nám oznámí, že je automat v pořádku. Následně vložíme slova, které si budeme chtít ověřit zda projdou automatem.

Na obrázku si můžeme všimnout, že je zvolená volba pro nedeterministický automat a opět jsem zadal všechny možné vstupy do tohoto automatu. Nastali zde všechny možné případy, vyjímkou nebyl ani vstup s neplatnými znaky.

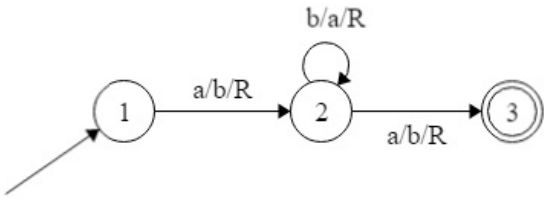


Obrázek 14: NKA - Test 1: Správně zadaný automat

Obrázek 15: NKA - Test 1: Výsledky

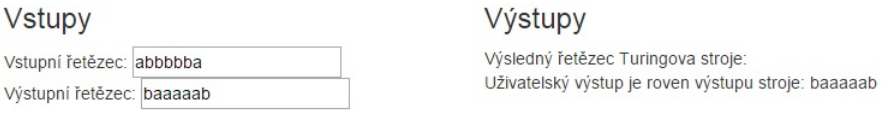
A.3 Turingovy stroje

Nakonec si vyzkoušíme Turingův stroj s reálnými daty. Vytvořím v návrháři velmi jednoduchý Turingův stroj, který první a poslední písmeno pokud bude "a"změní na písmeno "b". V návrháři vypadá Turingův stroj, který má tuto funkcionalitu takto.



Obrázek 16: TNG - Test 1: Správně zadaný stroj

Tento stroj je z matematického hlediska správně navrhnut a proto zadáme vstupní řetězec "abbbbba". Pokud dáme ověřit slova, tak nám tento stroj vypíše jak výsledek "baaaaab". Což si můžeme povšimnout také na obrázku s výsledky.



Obrázek 17: TNG - Test 1: Výsledky

Momentálně jsme vyzkoušeli všechny modely s reálnými daty a zjistili jsme, že program funguje dle stanoveného zadání.

B Ovládání aplikace

Ovládání aplikace je velice intuitivní a uživatel po prvním přečtení a shlédnutí aplikace by měl vědět, jak se tato aplikace ovládá. Nyní popíši ovládání jednotlivých částí, návrháře a poté nápovědu k zadávání dat pro námi navrhnutý automat. Všechny tyto informace budou umístěny také v aplikaci, aby byly uživateli hned po ruce. Aplikace se nám spustí pomocí programu Visual Studio a následně spuštění souboru Hlavni. Tímto souborem se dostaneme na hlavní stranu aplikace kde nalezneme návrhář a pole pro vstupní data.

B.1 Návrhář

Nejprve si popíšeme postup při kreslení automatu v návrháři. Ovládací prvky jsou následující

- Dvojklik
    1. vytvoření nového uzlu
    2. změna konečného stavu uzlu
- klávesa Shift
    1. vytvoření přechodů

Následně si popíšeme jednotlivé prvky a přesné vysázení komponent. Nejprve začneme s uzly, kde pomocí dvojkliku do kreslící plochy se nám vytvoří nový uzel. Po každém dvojkliku kdekoliv do kreslící plochy se nám vytvoří nový uzel, pokud chceme danému uzlu změnit stav na stav konečný, neboli také přijímací, opět dvojklikem na příslušný uzel.

Dále zde máme tvoření přechodů, přechody tvoříme pomocí stisknuté klávesy shift. A zde máme vytvoření tři možných přechodů.

- Smyčka přechod
- Start přechod
- Přechod z uzlu do uzlu

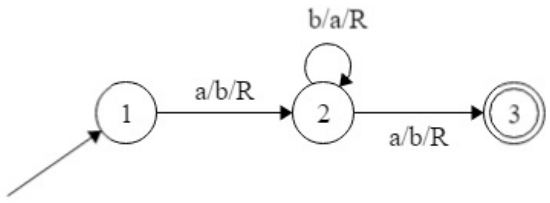
Smyčku přechod vytvoříme pomocí stisknu klávesy shift a následného kliku na příslušný uzel. Dále nesmíme opomenout pojmenování přechodu. Přechod typu start vytvoříme opět pomocí stisku klávesy shift a držení levého tlačítka myši z prázdného prostoru směrem do námi požadovaného startovního uzlu. U tohoto přechodu nezadáváme jméno přechodu, jelikož startovní přechod žádné nepotřebuje. Jako poslední, zde máme přechod z uzlu do uzlu. Tento přechod vytvoříme tak, že držíme klávesu shift a po vybrání námi požadovaného prvního uzlu stiskneme, držíme levé tlačítko myši a přetáhneme náš přechod do druhého, neboli následujícího uzlu. Opět nesmíme opomenout pojmenování přechodů, jelikož pro výpočet jsou tyto informace velmi důležité.

B.1.1 Data v návrháři

Zde si vysvětlíme zadávání dat v návrháři pro uzly a přechody. U automatů to není nic složitého a probíhá to stejně jako u kreslení grafu automatu na papíře. Jediné omezení zde je, že názvy uzlů a přechodů musí být vytvořeny z povolených znaků, což je malá, velká abeceda a číslice.

Složitější zadávání dat může nastat u Turingových strojů. Pojmenovávání uzlů probíhá stejně jako u automatů, kdy je název tvořen jedním znakem a to opět z povoleného rozsahu znaků. Pojmenovávání přechodů už musí splňovat určité pravidla. Text přechodu musí být ve specifickém formátu, přesněji *a,b/c/R*. Tento formát obsahuje všechny potřebné informace pro stroj a je ve formátu čti/zapiš/posuň se. První informace, což jsou znaky před prvním lomítkem obsahují tedy instrukce pro čtení. Z ukázky vidíme, že pokud stroj na vstupní pásce přečte znak "a"nebo "b", tak poté vykoná následující funkcionalitu, která se skrývá v textovém formátu. Druhá informace je tedy znak, který má na původní místo zapsat a v našem případě znak "c". V aplikaci je taky funkce, která při zapsání znaku **N** zanechá původní hodnotu a pouze se vykoná třetí informace. Poslední informace tedy slouží pro přechod po naší pomyslné pásce. Naší pomyslné hlavě můžeme určit, kterým směrem se pohne a to doleva, doprava nebo zůstane na místě. K tomuto nám slouží příkazy **S,L,R**, kdy znak S jako Stop nám značí, že zůstane na svém místě, L jako Left neboli doleva řekneme čtecí hlavě, že se posune doleva. Nakonec zde máme znak R jako Right neboli doprava a tímto přikážeme posun vpravo.

Na obrázku máme názornou ukázku namodelovaného Turingova stroje. Po přečtení podkapitoly bychom měli jednoduše rozpoznat a říci funkcionalitu stroje.



Obrázek 18: Zadávání dat do návrháře

B.2 Zadávání vstupních dat

V této podkapitole si vysvětlíme zadávání dat. Po námi vytvořeném automatu si zvolíme z nabídky, který výpočetní model jsme si navrhli a poté pomocí tlačítka Zkouška automatu vyzkoušíme, zda námi vytvořený automat splňuje všechny matematické náležitosti. Následně nám vypíše řetězec, zda vše souhlasí nebo zda máme někde chybu a určí typ chyby, kterou musíme opravit. Až po úspěšném navrhnutí automatu se nám odemkne tlačítko pro ověřování slov.

Dále nám nasledují vstupy pro aplikaci, které se liší pro dané modely. Pro automaty zde máme textové pole, kde vpisujeme vstupní slova. Nemusíme se obávat, že bychom

zadali špatné slovo. Program je dostatečně inteligentní a pokud slovo bude špatně nebo bude obsahovat nepovolené znaky, tak to oznámí uživateli. Slova v textovém poli oddělujeme pomocí klávesy enter. Po zadání všech námi požadovaných slov klikneme na tlačítko Ověřit slova, ve vedlejším sloupci se nám objeví výsledky pro všechny zadané slova.

Další možný výpočetní model je Turingův stroj, pro který máme trochu jiné vstupní požadavky. Zde máme dvě textová pole a to pro zadání vstupního řetězce, který má Turingův stroj zpracovat a poté textové pole pro zadání výstupního řetězce, podle kterého si student může ověřit, zda si ho navrhl správně a stroj dělá vše co má. Ve vedlejším sloupci nám aplikace oznámí, jestli námi napsaný výsledek odpovídá reálnému výsledku a reálný výsledek nám to vypíše také.